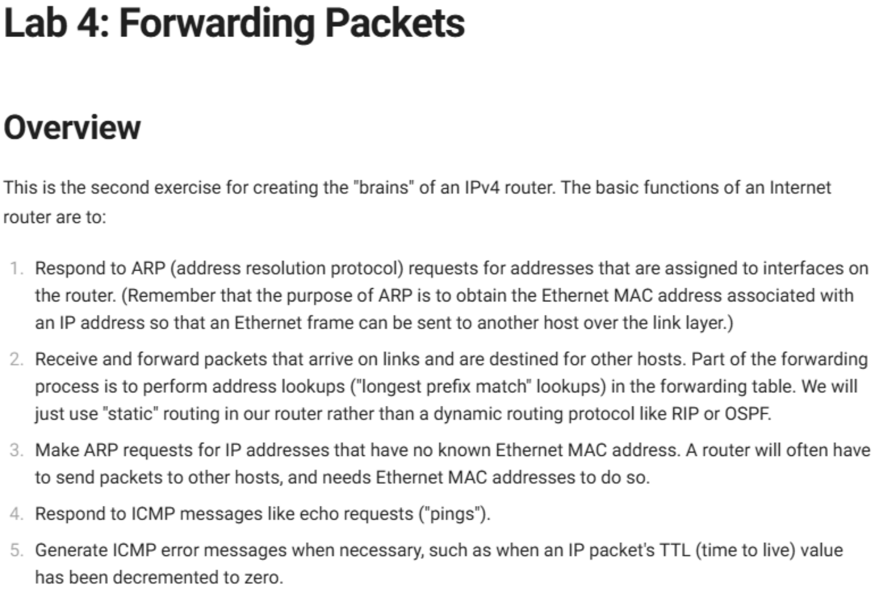
南京大学本科生实验报告

课程名称：计算机网络　　　　任课教师：田臣/李文中　　　　助教：

学院	计算机科学与技术	专业（方向）	计算机科学与技术
学号	**205220020**	姓名	金载润
Email	**jinzairun010227@naver.com**	开始/完成日期	**2024.10.25~2024.11.07**

1.实验名称



Lab 4: Forwarding Packets

Overview

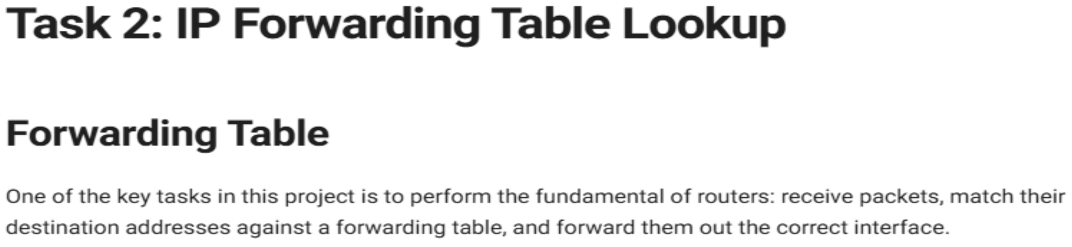
This is the second exercise for creating the "brains" of an IPv4 router. The basic functions of an Internet router are to:

1. Respond to ARP (address resolution protocol) requests for addresses that are assigned to interfaces on the router. (Remember that the purpose of ARP is to obtain the Ethernet MAC address associated with an IP address so that an Ethernet frame can be sent to another host over the link layer.)
2. Receive and forward packets that arrive on links and are destined for other hosts. Part of the forwarding process is to perform address lookups ("longest prefix match" lookups) in the forwarding table. We will just use "static" routing in our router rather than a dynamic routing protocol like RIP or OSPF.
3. Make ARP requests for IP addresses that have no known Ethernet MAC address. A router will often have to send packets to other hosts, and needs Ethernet MAC addresses to do so.
4. Respond to ICMP messages like echo requests ("pings").
5. Generate ICMP error messages when necessary, such as when an IP packet's TTL (time to live) value has been decremented to zero.

2. 实验目的

完成IPv4路由器的第一个功能：响应ARP包

3.实验内容 / 核心代码



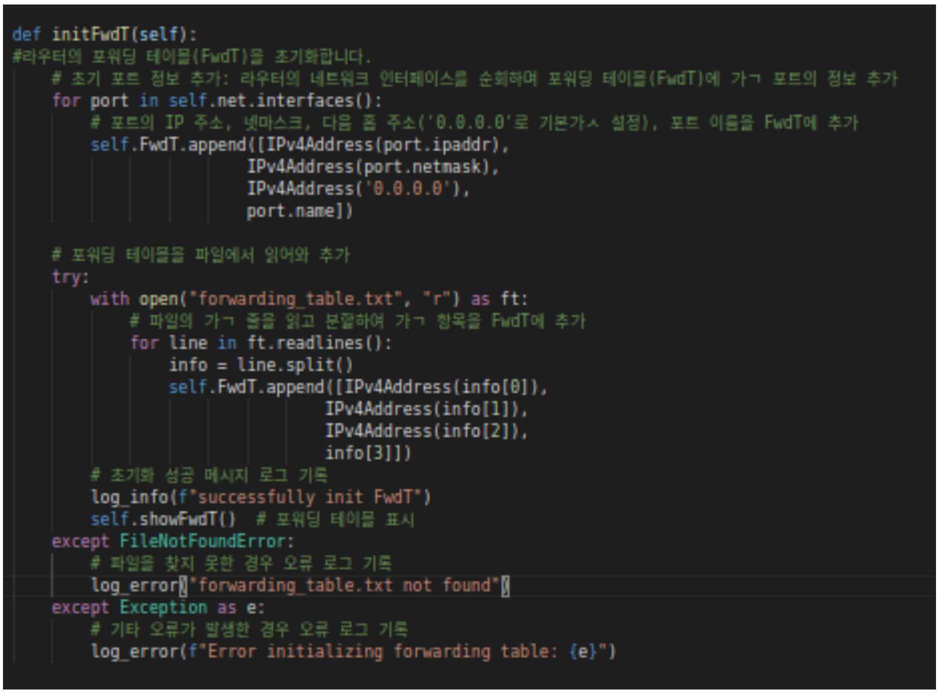
Task 2: IP Forwarding Table Lookup

Forwarding Table

One of the key tasks in this project is to perform the fundamental of routers: receive packets, match their destination addresses against a forwarding table, and forward them out the correct interface.

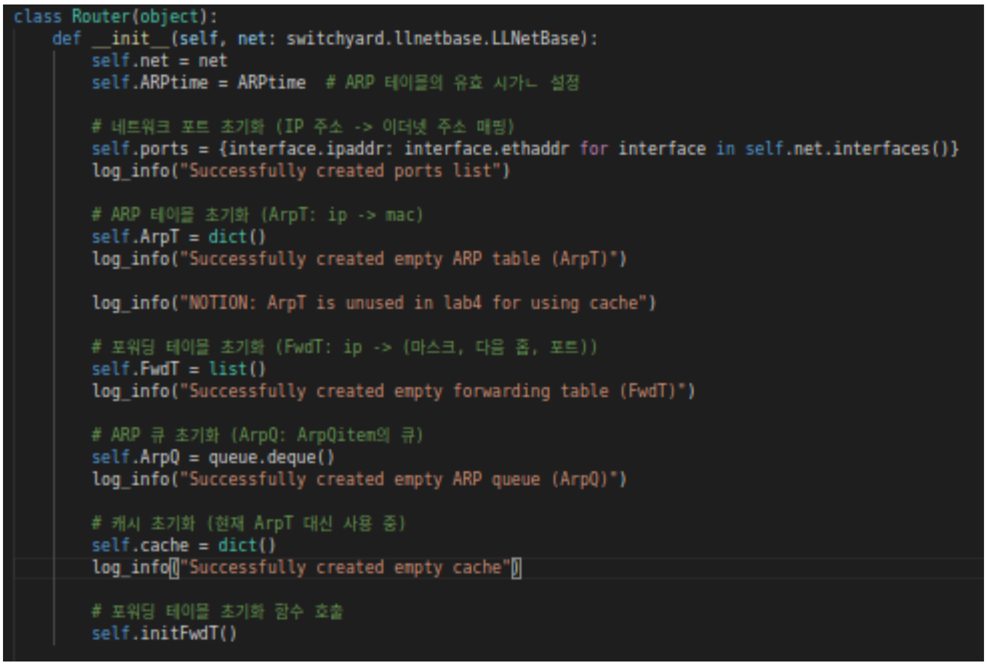
描述涉及到路由器和交换机在处理数据包时如何使用转发表和ARP缓存来优化包的转发和地址解析

转发表的搭建：



```
def initFwdT(self):
#라우터의 포워딩 테이블(FwdT)을 초기화합니다.
    # 초기 포트 정보 추가: 라우터의 네트워크 인터페이스를 순회하며 포워딩 테이블(FwdT)에 가ㄱ 포트의 정보 추가
    for port in self.net.interfaces():
        # 포트의 IP 주소, 넷마스크, 다음 홉 주소('0.0.0.0'로 기본가ㅅ 설정), 포트 이름을 FwdT에 추가
        self.FwdT.append([IPv4Address(port.ipaddr),
                        IPv4Address(port.netmask),
                        IPv4Address('0.0.0.0'),
                        port.name])

    # 포워딩 테이블을 파일에서 읽어와 추가
    try:
        with open("forwarding_table.txt", "r") as ft:
            # 파일의 가ㄱ 줄을 읽고 분할하여 가ㄱ 항목을 FwdT에 추가
            for line in ft.readlines():
                info = line.split()
                self.FwdT.append([IPv4Address(info[0]),
                                IPv4Address(info[1]),
                                IPv4Address(info[2]),
                                info[3]])
        # 초기화 성공 메시지 로그 기록
        log_info(f"successfully init FwdT")
        self.showFwdT()  # 포워딩 테이블 표시
    except FileNotFoundError:
        # 파일을 찾지 못한 경우 오류 로그 기록
        log_error("forwarding_table.txt not found")
    except Exception as e:
        # 기타 오류가 발생한 경우 오류 로그 기록
        log_error(f"Error initializing forwarding table: {e}")
```

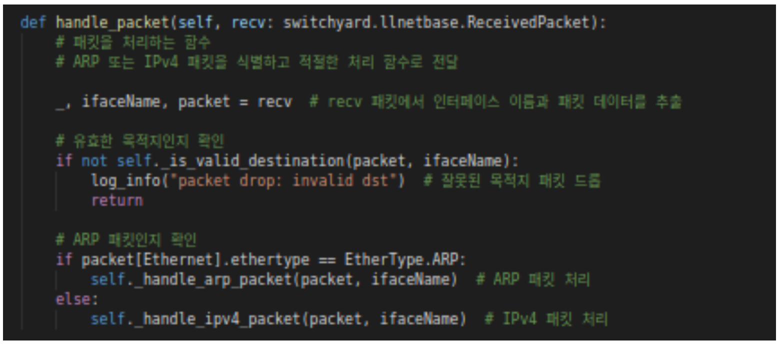
ARP地址解析表：【在本次实验中，转发表是不会更新的；但是ARP地址解析表应当保留，否则路由器无法学习IP-MAC对应关系， 将总是发送ARP请求以获取MAC地址。这一任务可以通过改造lab3的ARP表来实现】



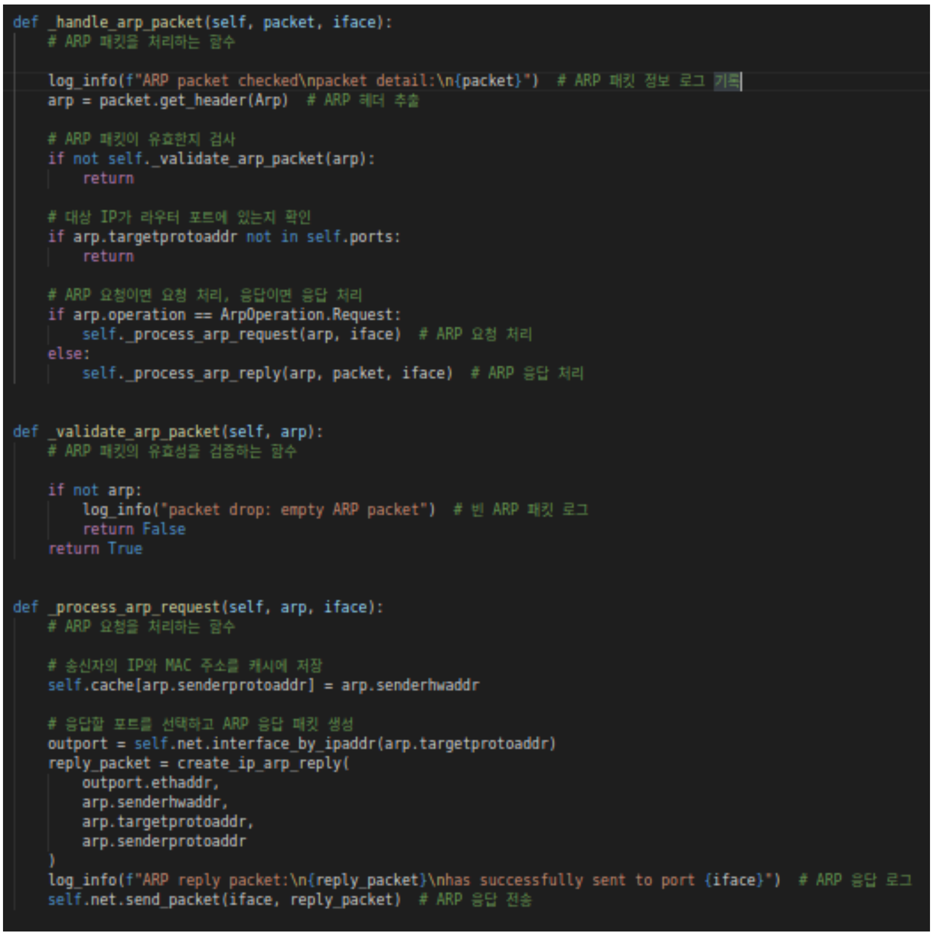
```
class Router(object):
    def __init__(self, net: switchyard.llnetbase.LLNetBase):
        self.net = net
        self.ARPtime = ARPtime  # ARP 테이블의 유효 시가ㄴ 설정

        # 네트워크 포트 초기화 (IP 주소 -> 이더넷 주소 매핑)
        self.ports = {interface.ipaddr: interface.ethaddr for interface in self.net.interfaces()}
        log_info("Successfully created ports list")

        # ARP 테이블 초기화 (ArpT: ip -> mac)
        self.ArpT = dict()
        log_info("Successfully created empty ARP table (ArpT)")

        log_info("NOTION: ArpT is unused in lab4 for using cache")

        # 포워딩 테이블 초기화 (FwdT: ip -> (마스크, 다음 홉, 포트))
        self.FwdT = list()
        log_info("Successfully created empty forwarding table (FwdT)")

        # ARP 큐 초기화 (ArpQ: ArpQitem의 큐)
        self.ArpQ = queue.deque()
        log_info("Successfully created empty ARP queue (ArpQ)")

        # 캐시 초기화 (현재 ArpT 대신 사용 중)
        self.cache = dict()
        log_info("Successfully created empty cache")

        # 포워딩 테이블 초기화 함수 호출
        self.initFwdT()
```

学习地址解析信息, 数据包转发逻辑即为 handle_packet 函数的实现方案：

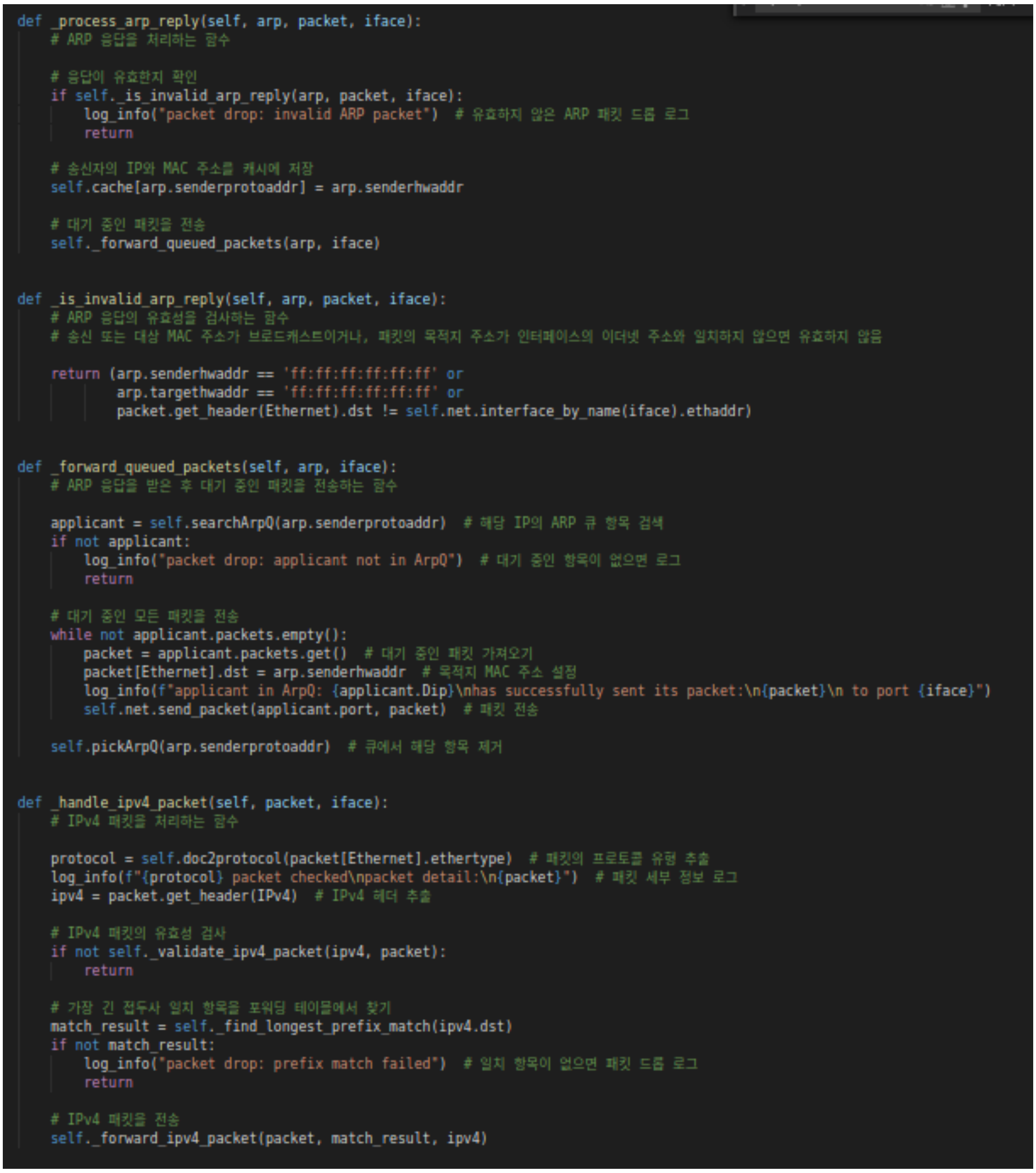
下面为帮助handle_packet()的函数



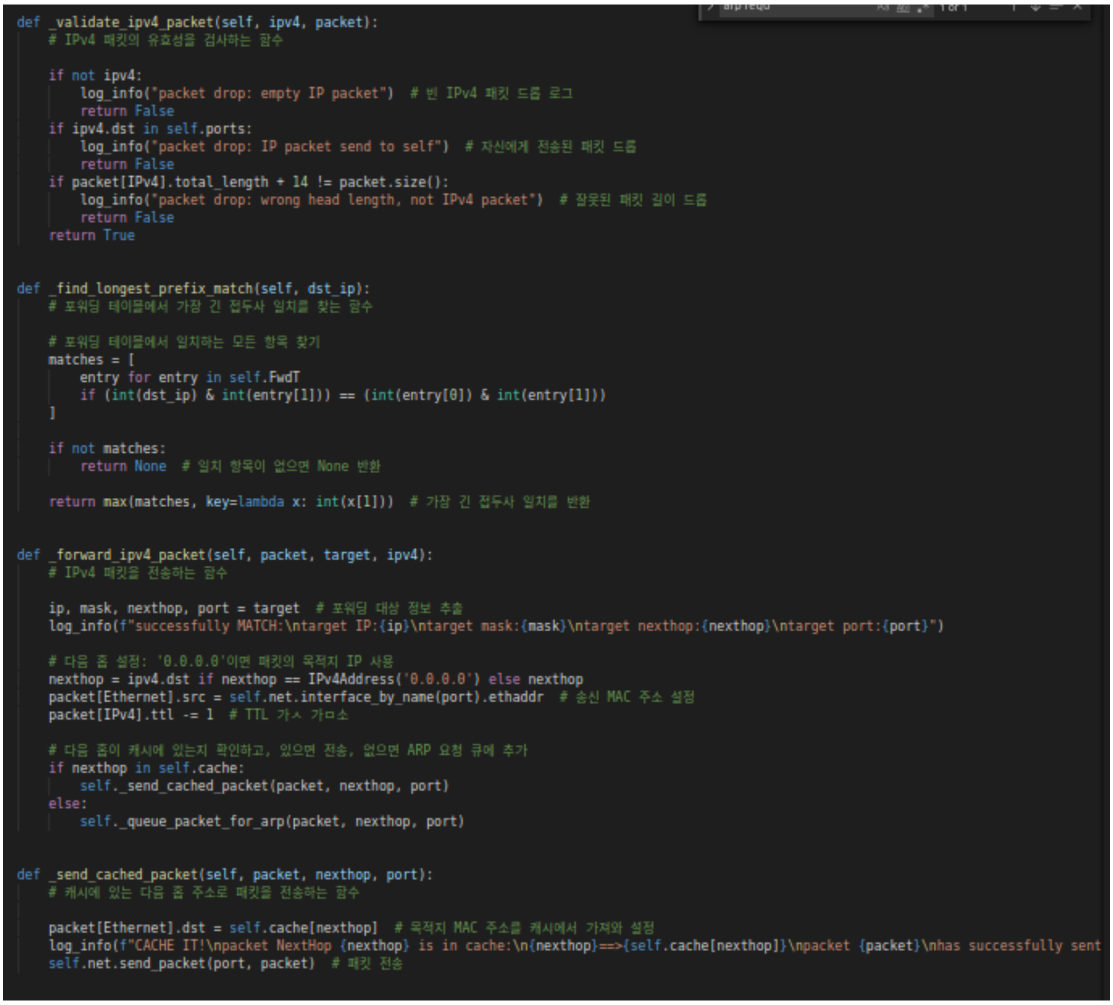
```python
def handle_packet(self, recv: switchyard.llnetbase.ReceivedPacket):
    # 패킷을 처리하는 함수
    # ARP 또는 IPv4 패킷을 식별하고 적절한 처리 함수로 전달

    _, ifaceName, packet = recv  # recv 패킷에서 인터페이스 이름과 패킷 데이터를 추출

    # 유효한 목적지인지 확인
    if not self._is_valid_destination(packet, ifaceName):
        log_info("packet drop: invalid dst")  # 잘못된 목적지 패킷 드롭
        return

    # ARP 패킷인지 확인
    if packet[Ethernet].ethertype == EtherType.ARP:
        self._handle_arp_packet(packet, ifaceName)  # ARP 패킷 처리
    else:
        self._handle_ipv4_packet(packet, ifaceName)  # IPv4 패킷 처리
```



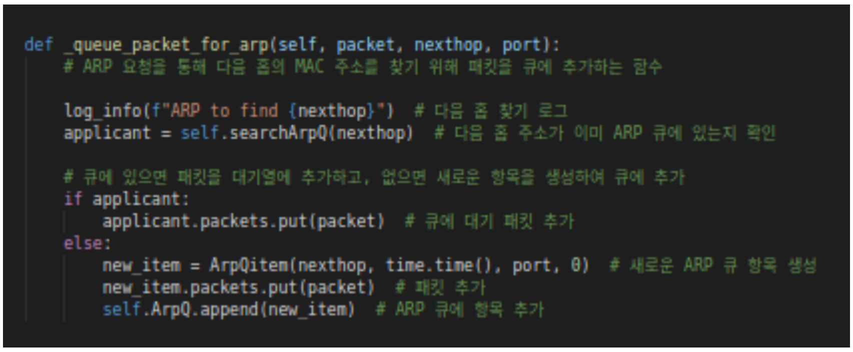
```python
def _handle_arp_packet(self, packet, iface):
    # ARP 패킷을 처리하는 함수

    log_info(f"ARP packet checked\npacket detail:\n{packet}")  # ARP 패킷 정보 로그 기록
    arp = packet.get_header(Arp)  # ARP 헤더 추출

    # ARP 패킷이 유효한지 검사
    if not self._validate_arp_packet(arp):
        return

    # 대상 IP가 라우터 포트에 있는지 확인
    if arp.targetprotoaddr not in self.ports:
        return

    # ARP 요청이면 요청 처리, 응답이면 응답 처리
    if arp.operation == ArpOperation.Request:
        self._process_arp_request(arp, iface)  # ARP 요청 처리
    else:
        self._process_arp_reply(arp, packet, iface)  # ARP 응답 처리


def _validate_arp_packet(self, arp):
    # ARP 패킷의 유효성을 검증하는 함수

    if not arp:
        log_info("packet drop: empty ARP packet")  # 빈 ARP 패킷 로그
        return False
    return True


def _process_arp_request(self, arp, iface):
    # ARP 요청을 처리하는 함수

    # 송신자의 IP와 MAC 주소를 캐시에 저장
    self.cache[arp.senderprotoaddr] = arp.senderhwaddr

    # 응답할 포트를 선택하고 ARP 응답 패킷 생성
    outport = self.net.interface_by_ipaddr(arp.targetprotoaddr)
    reply_packet = create_ip_arp_reply(
        outport.ethaddr,
        arp.senderhwaddr,
        arp.targetprotoaddr,
        arp.senderprotoaddr
    )
    log_info(f"ARP reply packet:\n{reply_packet}\nhas successfully sent to port {iface}")  # ARP 응답 로그
    self.net.send_packet(iface, reply_packet)  # ARP 응답 전송
```



```python
def _process_arp_reply(self, arp, packet, iface):
    # ARP 응답을 처리하는 함수

    # 응답이 유효한지 확인
    if self._is_invalid_arp_reply(arp, packet, iface):
        log_info("packet drop: invalid ARP packet")  # 유효하지 않은 ARP 패킷 드롭 로그
        return

    # 송신자의 IP와 MAC 주소를 캐시에 저장
    self.cache[arp.senderprotoaddr] = arp.senderhwaddr

    # 대기 중인 패킷을 전송
    self._forward_queued_packets(arp, iface)


def _is_invalid_arp_reply(self, arp, packet, iface):
    # ARP 응답의 유효성을 검사하는 함수
    # 송신 또는 대상 MAC 주소가 브로드캐스트이거나, 패킷의 목적지 주소가 인터페이스의 이더넷 주소와 일치하지 않으면 유효하지 않음

    return (arp.senderhwaddr == 'ff:ff:ff:ff:ff:ff' or
            arp.targethwaddr == 'ff:ff:ff:ff:ff:ff' or
            packet.get_header(Ethernet).dst != self.net.interface_by_name(iface).ethaddr)


def _forward_queued_packets(self, arp, iface):
    # ARP 응답을 받은 후 대기 중인 패킷을 전송하는 함수

    applicant = self.searchArpQ(arp.senderprotoaddr)  # 해당 IP의 ARP 큐 항목 검색
    if not applicant:
        log_info("packet drop: applicant not in ArpQ")  # 대기 중인 항목이 없으면 로그
        return

    # 대기 중인 모든 패킷을 전송
    while not applicant.packets.empty():
        packet = applicant.packets.get()  # 대기 중인 패킷 가져오기
        packet[Ethernet].dst = arp.senderhwaddr  # 목적지 MAC 주소 설정
        log_info(f"applicant in ArpQ: {applicant.Dip}\nhas successfully sent its packet:\n{packet}\n to port {iface}")
        self.net.send_packet(applicant.port, packet)  # 패킷 전송

    self.pickArpQ(arp.senderprotoaddr)  # 큐에서 해당 항목 제거


def _handle_ipv4_packet(self, packet, iface):
    # IPv4 패킷을 처리하는 함수

    protocol = self.doc2protocol(packet[Ethernet].ethertype)  # 패킷의 프로토콜 유형 추출
    log_info(f"{protocol} packet checked\npacket detail:\n{packet}")  # 패킷 세부 정보 로그
    ipv4 = packet.get_header(IPv4)  # IPv4 헤더 추출

    # IPv4 패킷의 유효성 검사
    if not self._validate_ipv4_packet(ipv4, packet):
        return

    # 가장 긴 접두사 일치 항목을 포워딩 테이블에서 찾기
    match_result = self._find_longest_prefix_match(ipv4.dst)
    if not match_result:
        log_info("packet drop: prefix match failed")  # 일치 항목이 없으면 패킷 드롭 로그
        return

    # IPv4 패킷을 전송
    self._forward_ipv4_packet(packet, match_result, ipv4)
```
]



```python
def _validate_ipv4_packet(self, ipv4, packet):
    # IPv4 패킷의 유효성을 검사하는 함수

    if not ipv4:
        log_info("packet drop: empty IP packet")  # 빈 IPv4 패킷 드롭 로그
        return False
    if ipv4.dst in self.ports:
        log_info("packet drop: IP packet send to self")  # 자신에게 전송된 패킷 드롭
        return False
    if packet[IPv4].total_length + 14 != packet.size():
        log_info("packet drop: wrong head length, not IPv4 packet")  # 잘못된 패킷 길이 드롭
        return False
    return True


def _find_longest_prefix_match(self, dst_ip):
    # 포워딩 테이블에서 가장 긴 접두사 일치를 찾는 함수

    # 포워딩 테이블에서 일치하는 모든 항목 찾기
    matches = [
        entry for entry in self.FwdT
        if (int(dst_ip) & int(entry[1])) == (int(entry[0]) & int(entry[1]))
    ]

    if not matches:
        return None  # 일치 항목이 없으면 None 반환

    return max(matches, key=lambda x: int(x[1]))  # 가장 긴 접두사 일치를 반환


def _forward_ipv4_packet(self, packet, target, ipv4):
    # IPv4 패킷을 전송하는 함수

    ip, mask, nexthop, port = target  # 포워딩 대상 정보 추출
    log_info(f"successfully MATCH:\ntarget IP:{ip}\ntarget mask:{mask}\ntarget nexthop:{nexthop}\ntarget port:{port}")

    # 다음 홉 설정: '0.0.0.0'이면 패킷의 목적지 IP 사용
    nexthop = ipv4.dst if nexthop == IPv4Address('0.0.0.0') else nexthop
    packet[Ethernet].src = self.net.interface_by_name(port).ethaddr  # 송신 MAC 주소 설정
    packet[IPv4].ttl -= 1  # TTL 가ㅅ 가ㅁ소

    # 다음 홉이 캐시에 있는지 확인하고, 있으면 전송, 없으면 ARP 요청 큐에 추가
    if nexthop in self.cache:
        self._send_cached_packet(packet, nexthop, port)
    else:
        self._queue_packet_for_arp(packet, nexthop, port)


def _send_cached_packet(self, packet, nexthop, port):
    # 캐시에 있는 다음 홉 주소로 패킷을 전송하는 함수

    packet[Ethernet].dst = self.cache[nexthop]  # 목적지 MAC 주소를 캐시에서 가져와 설정
    log_info(f"CACHE IT!\npacket NextHop {nexthop} is in cache:\n{nexthop}==>{self.cache[nexthop]}\npacket {packet}\nhas successfully sent
    self.net.send_packet(port, packet)  # 패킷 전송
```



```python
def _queue_packet_for_arp(self, packet, nexthop, port):
    # ARP 요청을 통해 다음 홉의 MAC 주소를 찾기 위해 패킷을 큐에 추가하는 함수

    log_info(f"ARP to find {nexthop}")  # 다음 홉 찾기 로그
    applicant = self.searchArpQ(nexthop)  # 다음 홉 주소가 이미 ARP 큐에 있는지 확인

    # 큐에 있으면 패킷을 대기열에 추가하고, 없으면 새로운 항목을 생성하여 큐에 추가
    if applicant:
        applicant.packets.put(packet)  # 큐에 대기 패킷 추가
    else:
        new_item = ArpQitem(nexthop, time.time(), port, 0)  # 새로운 ARP 큐 항목 생성
        new_item.packets.put(packet)  # 패킷 추가
        self.ArpQ.append(new_item)  # ARP 큐에 항목 추가
```

Task 3: Forwarding the Packet and ARP

Send ARP Request and Forward Packet

After you lookup an IP destination address, the next steps are to:

1. Decrement the TTL field in the IP header by 1. You can assume for this project that the TTL value is greater than 0 after decrementing. We'll handle "expired" TTLs in Lab 5.
2. Create a new Ethernet header for the IP packet to be forwarded. To construct the Ethernet header, you need to know the destination Ethernet MAC address corresponding to the host to which the packet should be forwarded. The next hop host is either:
    a. the destination host, if the destination address is directly reachable through one of the router interfaces (i.e., the subnet that the destination address belongs to is directly connected to a router interface), or
    b. it is an IP address on a router through which the destination is reachable.

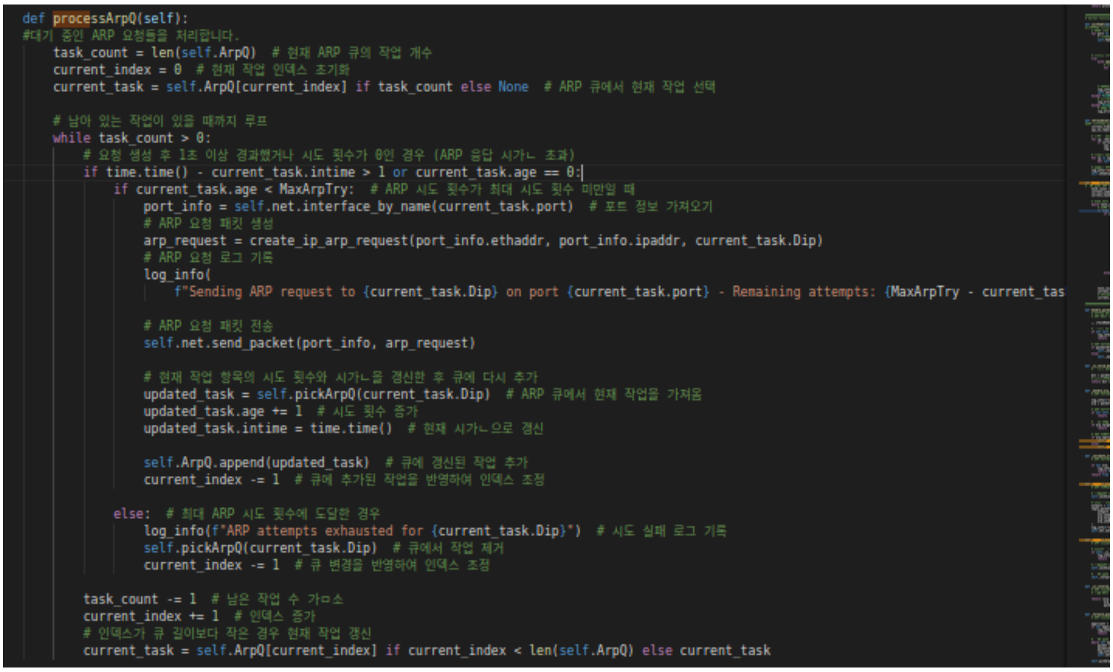
在步骤1中，我们已经创建了转发表，维护了ARP缓存，并完成了包处理的逻辑。接下来需要实现转发功能，这是一个涉及时序的任务。交换机的工作流程应该是这样的：收到包 ⇒ 处理包 ⇒ 等待处理ARP请求队列。在处理ARP请求队列时，需要按照先进先出的顺序逐一处理每个请求。对于每个请求，首先检查它是第几次发起请求：

*如果是第一次发起请求，我们不应该让它等待超过1秒，而是立即发起请求，并将该请求移到队列的末尾。

*如果不是第一次，但请求次数还不到5次，我们将等待1秒后再次发起请求，并将该请求移到队列的末尾。

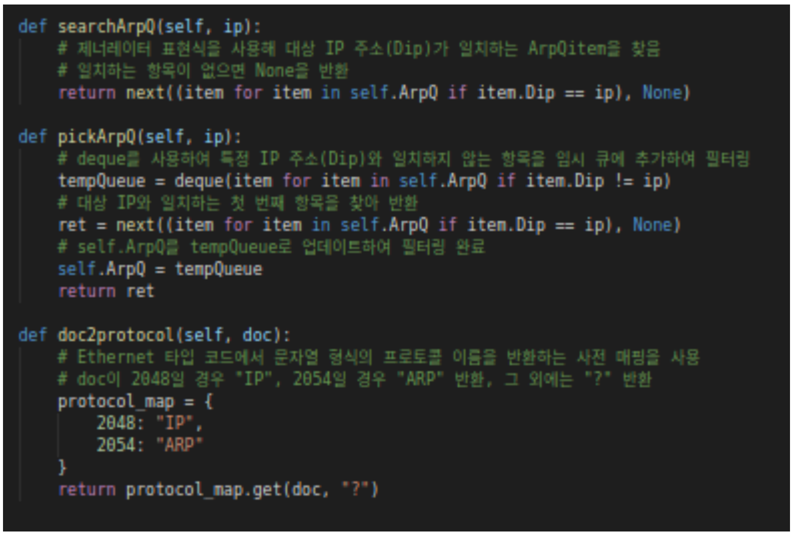
*如果请求次数已满5次，则直接从队列中删除该请求。

为了实现这一处理逻辑，我们将在一个while循环中反复调用处理函数，直到收到包为止。在每轮循环中，逐个访问队列中的所有请求，并确保每个请求仅被访问一次。



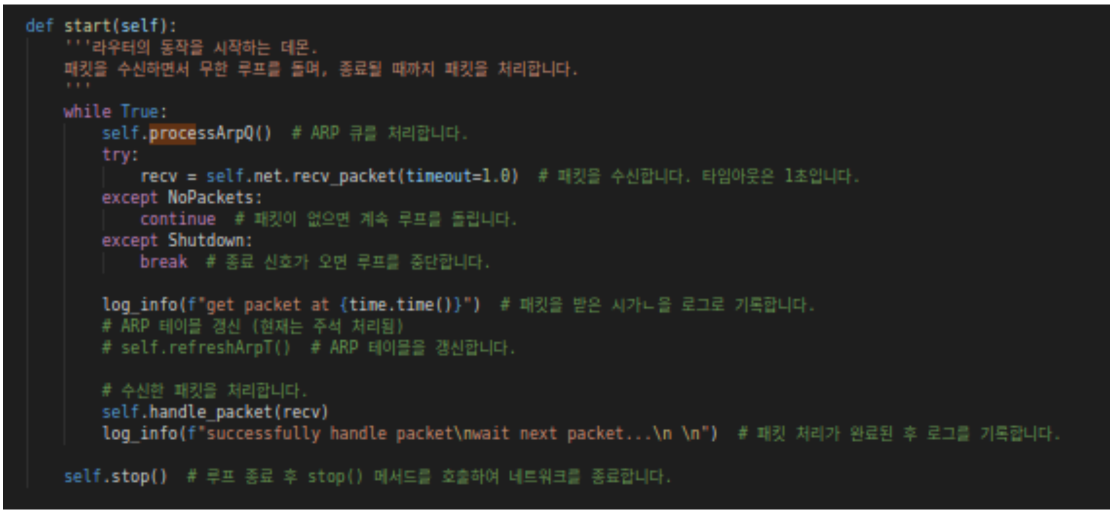
```
def processArpQ(self):
#대기 중인 ARP 요청들을 처리합니다.
    task_count = len(self.ArpQ)  # 현재 ARP 큐의 작업 개수
    current_index = 0  # 현재 작업 인덱스 초기화
    current_task = self.ArpQ[current_index] if task_count else None  # ARP 큐에서 현재 작업 선택

    # 남아 있는 작업이 있을 때까지 루프
    while task_count > 0:
        # 요청 생성 후 1초 이상 경과했거나 시도 횟수가 0인 경우 (ARP 응답 시가ㄴ 초과)
        if time.time() - current_task.intime > 1 or current_task.age == 0:
            if current_task.age < MaxArpTry:  # ARP 시도 횟수가 최대 시도 횟수 미만일 때
                port_info = self.net.interface_by_name(current_task.port)  # 포트 정보 가져오기
                # ARP 요청 패킷 생성
                arp_request = create_ip_arp_request(port_info.ethaddr, port_info.ipaddr, current_task.Dip)
                # ARP 요청 로그 기록
                log_info(
                    f"Sending ARP request to {current_task.Dip} on port {current_task.port} - Remaining attempts: {MaxArpTry - current_tas

                # ARP 요청 패킷 전송
                self.net.send_packet(port_info, arp_request)

                # 현재 작업 항목의 시도 횟수와 시가ㄴ을 갱신한 후 큐에 다시 추가
                updated_task = self.pickArpQ(current_task.Dip)  # ARP 큐에서 현재 작업을 가져옴
                updated_task.age += 1  # 시도 횟수 증가
                updated_task.intime = time.time()  # 현재 시가ㄴ으로 갱신

                self.ArpQ.append(updated_task)  # 큐에 갱신된 작업 추가
                current_index -= 1  # 큐에 추가된 작업을 반영하여 인덱스 조정

            else:  # 최대 ARP 시도 횟수에 도달한 경우
                log_info(f"ARP attempts exhausted for {current_task.Dip}")  # 시도 실패 로그 기록
                self.pickArpQ(current_task.Dip)  # 큐에서 작업 제거
                current_index -= 1  # 큐 변경을 반영하여 인덱스 조정

        task_count -= 1  # 남은 작업 수 가ㅁ소
        current_index += 1  # 인덱스 증가
        # 인덱스가 큐 길이보다 작은 경우 현재 작업 갱신
        current_task = self.ArpQ[current_index] if current_index < len(self.ArpQ) else current_task
```

帮助函数:



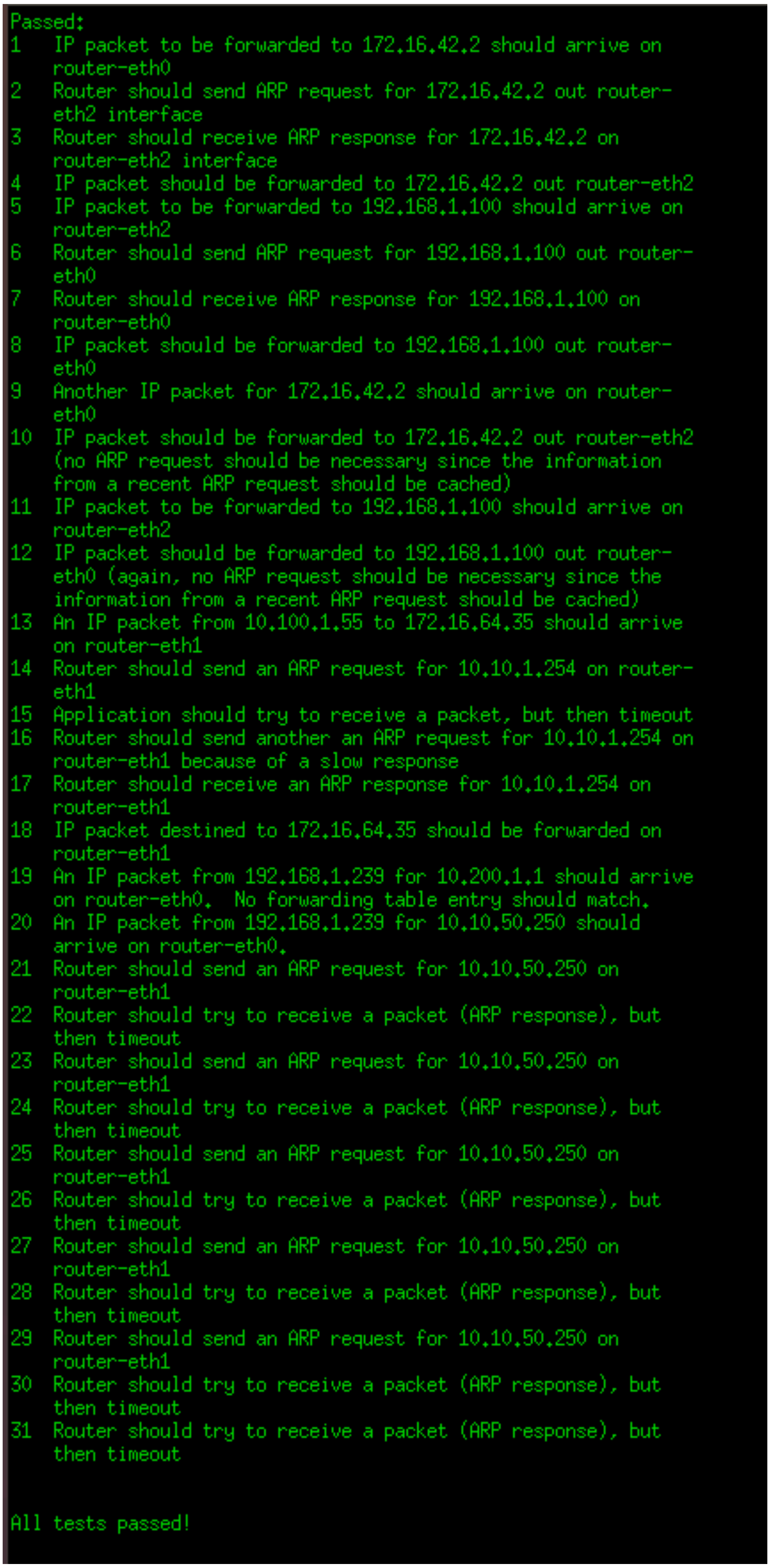
```python
def searchArpQ(self, ip):
    # 제너레이터 표현식을 사용해 대상 IP 주소(Dip)가 일치하는 ArpQitem을 찾음
    # 일치하는 항목이 없으면 None을 반환
    return next((item for item in self.ArpQ if item.Dip == ip), None)

def pickArpQ(self, ip):
    # deque를 사용하여 특정 IP 주소(Dip)와 일치하지 않는 항목을 임시 큐에 추가하여 필터링
    tempQueue = deque(item for item in self.ArpQ if item.Dip != ip)
    # 대상 IP와 일치하는 첫 번째 항목을 찾아 반환
    ret = next((item for item in self.ArpQ if item.Dip == ip), None)
    # self.ArpQ를 tempQueue로 업데이트하여 필터링 완료
    self.ArpQ = tempQueue
    return ret

def doc2protocol(self, doc):
    # Ethernet 타입 코드에서 문자열 형식의 프로토콜 이름을 반환하는 사전 매핑을 사용
    # doc이 2048일 경우 "IP", 2054일 경우 "ARP" 반환, 그 외에는 "?" 반환
    protocol_map = {
        2048: "IP",
        2054: "ARP"
    }
    return protocol_map.get(doc, "?")
```

最终，在主逻辑中添加队列处理函数：

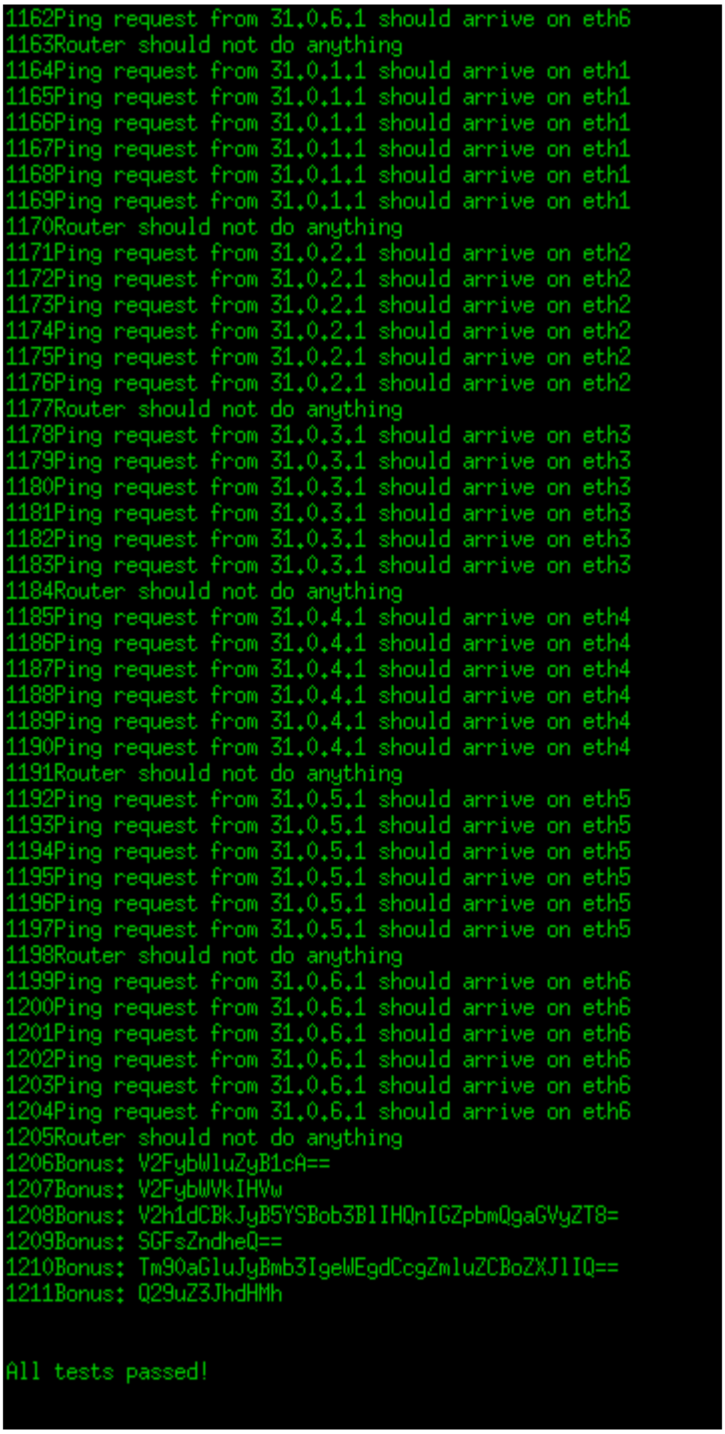


```python
def start(self):
    '''라우터의 동작을 시작하는 데몬.
    패킷을 수신하면서 무한 루프를 돌며, 종료될 때까지 패킷을 처리합니다.
    '''
    while True:
        self.processArpQ()  # ARP 큐를 처리합니다.
        try:
            recv = self.net.recv_packet(timeout=1.0)  # 패킷을 수신합니다. 타임아웃은 1초입니다.
        except NoPackets:
            continue  # 패킷이 없으면 계속 루프를 돌립니다.
        except Shutdown:
            break  # 종료 신호가 오면 루프를 중단합니다.

        log_info(f"get packet at {time.time()}")  # 패킷을 받은 시가ㄴ을 로그로 기록합니다.
        # ARP 테이블 갱신 (현재는 주석 처리됨)
        # self.refreshArpT()  # ARP 테이블을 갱신합니다.

        # 수신한 패킷을 처리합니다.
        self.handle_packet(recv)
        log_info(f"successfully handle packet\nwait next packet...\n \n")  # 패킷 처리가 완료된 후 로그를 기록합니다.

    self.stop()  # 루프 종료 후 stop() 메서드를 호출하여 네트워크를 종료합니다.
```

4.实验结果



```
Passed:
1   IP packet to be forwarded to 172.16.42.2 should arrive on
    router-eth0
2   Router should send ARP request for 172.16.42.2 out router-
    eth2 interface
3   Router should receive ARP response for 172.16.42.2 on
    router-eth2 interface
4   IP packet should be forwarded to 172.16.42.2 out router-eth2
5   IP packet to be forwarded to 192.168.1.100 should arrive on
    router-eth2
6   Router should send ARP request for 192.168.1.100 out router-
    eth0
7   Router should receive ARP response for 192.168.1.100 on
    router-eth0
8   IP packet should be forwarded to 192.168.1.100 out router-
    eth0
9   Another IP packet for 172.16.42.2 should arrive on router-
    eth0
10  IP packet should be forwarded to 172.16.42.2 out router-eth2
    (no ARP request should be necessary since the information
    from a recent ARP request should be cached)
11  IP packet to be forwarded to 192.168.1.100 should arrive on
    router-eth2
12  IP packet should be forwarded to 192.168.1.100 out router-
    eth0 (again, no ARP request should be necessary since the
    information from a recent ARP request should be cached)
13  An IP packet from 10.100.1.55 to 172.16.64.35 should arrive
    on router-eth1
14  Router should send an ARP request for 10.10.1.254 on router-
    eth1
15  Application should try to receive a packet, but then timeout
16  Router should send another an ARP request for 10.10.1.254 on
    router-eth1 because of a slow response
17  Router should receive an ARP response for 10.10.1.254 on
    router-eth1
18  IP packet destined to 172.16.64.35 should be forwarded on
    router-eth1
19  An IP packet from 192.168.1.239 for 10.200.1.1 should arrive
    on router-eth0.  No forwarding table entry should match.
20  An IP packet from 192.168.1.239 for 10.10.50.250 should
    arrive on router-eth0.
21  Router should send an ARP request for 10.10.50.250 on
    router-eth1
22  Router should try to receive a packet (ARP response), but
    then timeout
23  Router should send an ARP request for 10.10.50.250 on
    router-eth1
24  Router should try to receive a packet (ARP response), but
    then timeout
25  Router should send an ARP request for 10.10.50.250 on
    router-eth1
26  Router should try to receive a packet (ARP response), but
    then timeout
27  Router should send an ARP request for 10.10.50.250 on
    router-eth1
28  Router should try to receive a packet (ARP response), but
    then timeout
29  Router should send an ARP request for 10.10.50.250 on
    router-eth1
30  Router should try to receive a packet (ARP response), but
    then timeout
31  Router should try to receive a packet (ARP response), but
    then timeout


All tests passed!
```



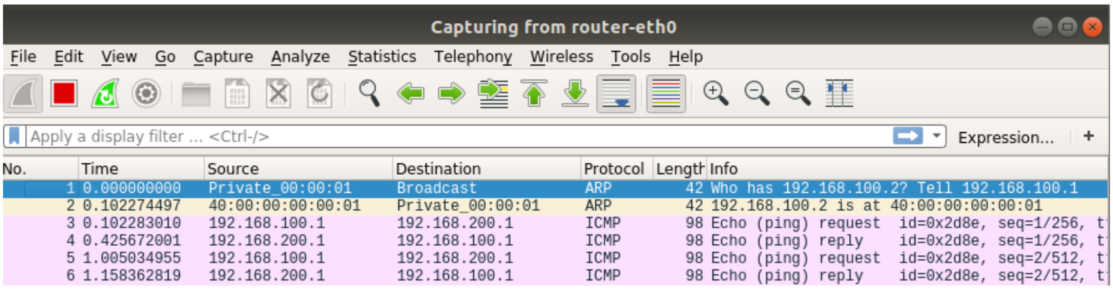
```
1162Ping request from 31.0.6.1 should arrive on eth6
1163Router should not do anything
1164Ping request from 31.0.1.1 should arrive on eth1
1165Ping request from 31.0.1.1 should arrive on eth1
1166Ping request from 31.0.1.1 should arrive on eth1
1167Ping request from 31.0.1.1 should arrive on eth1
1168Ping request from 31.0.1.1 should arrive on eth1
1169Ping request from 31.0.1.1 should arrive on eth1
1170Router should not do anything
1171Ping request from 31.0.2.1 should arrive on eth2
1172Ping request from 31.0.2.1 should arrive on eth2
1173Ping request from 31.0.2.1 should arrive on eth2
1174Ping request from 31.0.2.1 should arrive on eth2
1175Ping request from 31.0.2.1 should arrive on eth2
1176Ping request from 31.0.2.1 should arrive on eth2
1177Router should not do anything
1178Ping request from 31.0.3.1 should arrive on eth3
1179Ping request from 31.0.3.1 should arrive on eth3
1180Ping request from 31.0.3.1 should arrive on eth3
1181Ping request from 31.0.3.1 should arrive on eth3
1182Ping request from 31.0.3.1 should arrive on eth3
1183Ping request from 31.0.3.1 should arrive on eth3
1184Router should not do anything
1185Ping request from 31.0.4.1 should arrive on eth4
1186Ping request from 31.0.4.1 should arrive on eth4
1187Ping request from 31.0.4.1 should arrive on eth4
1188Ping request from 31.0.4.1 should arrive on eth4
1189Ping request from 31.0.4.1 should arrive on eth4
1190Ping request from 31.0.4.1 should arrive on eth4
1191Router should not do anything
1192Ping request from 31.0.5.1 should arrive on eth5
1193Ping request from 31.0.5.1 should arrive on eth5
1194Ping request from 31.0.5.1 should arrive on eth5
1195Ping request from 31.0.5.1 should arrive on eth5
1196Ping request from 31.0.5.1 should arrive on eth5
1197Ping request from 31.0.5.1 should arrive on eth5
1198Router should not do anything
1199Ping request from 31.0.6.1 should arrive on eth6
1200Ping request from 31.0.6.1 should arrive on eth6
1201Ping request from 31.0.6.1 should arrive on eth6
1202Ping request from 31.0.6.1 should arrive on eth6
1203Ping request from 31.0.6.1 should arrive on eth6
1204Ping request from 31.0.6.1 should arrive on eth6
1205Router should not do anything
1206Bonus: V2FybWluZyB1cA==
1207Bonus: V2FybWVkIHVw
1208Bonus: V2h1dCBkJyB5YSBob3BlIHQnIGZpbmQgaGVyZT8=
1209Bonus: SGFsZndheQ==
1210Bonus: Tm90aGluJyBmb3IgeWEgdCcgZmluZCBoZXJlIQ==
1211Bonus: Q29uZ3JhdHMh


All tests passed!
```

5.抓包测结果 ／ 解释

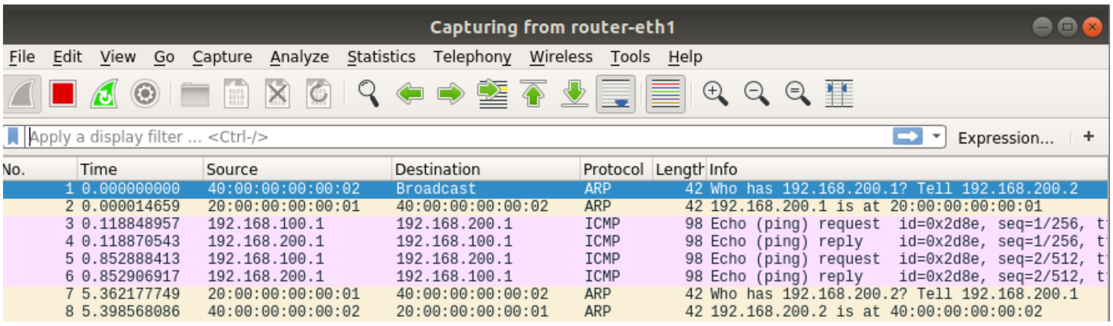
运行交换机后，检测 eth0 与 eth1

server1 ping -c2 server2

抓包结果如下：


Capturing from router-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
+

```
No.  Time         Source             Destination        Protocol Length Info
   1 0.000000000  Private_00:00:01   Broadcast          ARP        42 Who has 192.168.100.2? Tell 192.168.100.1
   2 0.102274497  40:00:00:00:00:01  Private_00:00:01   ARP        42 192.168.100.2 is at 40:00:00:00:00:01
   3 0.102283010  192.168.100.1      192.168.200.1      ICMP       98 Echo (ping) request  id=0x2d8e, seq=1/256, t
   4 0.425672001  192.168.200.1      192.168.100.1      ICMP       98 Echo (ping) reply    id=0x2d8e, seq=1/256, t
   5 1.005034955  192.168.100.1      192.168.200.1      ICMP       98 Echo (ping) request  id=0x2d8e, seq=2/512, t
   6 1.158362819  192.168.200.1      192.168.100.1      ICMP       98 Echo (ping) reply    id=0x2d8e, seq=2/512, t
```


Capturing from router-eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…
+

```
No.  Time         Source             Destination        Protocol Length Info
   1 0.000000000  40:00:00:00:00:02  Broadcast          ARP        42 Who has 192.168.200.1? Tell 192.168.200.2
   2 0.000014659  20:00:00:00:00:01  40:00:00:00:00:02  ARP        42 192.168.200.1 is at 20:00:00:00:00:01
   3 0.118848957  192.168.100.1      192.168.200.1      ICMP       98 Echo (ping) request  id=0x2d8e, seq=1/256, t
   4 0.118870543  192.168.200.1      192.168.100.1      ICMP       98 Echo (ping) reply    id=0x2d8e, seq=1/256, t
   5 0.852888413  192.168.100.1      192.168.200.1      ICMP       98 Echo (ping) request  id=0x2d8e, seq=2/512, t
   6 0.852906917  192.168.200.1      192.168.100.1      ICMP       98 Echo (ping) reply    id=0x2d8e, seq=2/512, t
   7 5.362177749  20:00:00:00:00:01  40:00:00:00:00:02  ARP        42 Who has 192.168.200.2? Tell 192.168.200.1
   8 5.398568086  40:00:00:00:00:02  20:00:00:00:00:01  ARP        42 192.168.200.2 is at 40:00:00:00:00:02
```

server1 需要向通往 server2 的路由器端口发送包，首先要获悉后者的MAC地址，因此server1 发起ARP广播，被 router 捕获后原路给出回答，此即 eth0 的前两个包

server1 掌握MAC地址后，向其发送目的为 server2 的ping request，该包被 router 捕获，此即 eth0 的第三个包

router 尚不清楚 server2 的MAC地址，但知道其在哪个端口，因此缓存ping包，从对应端口（40:00:00:00:00:02）广播询问 server2 的MAC地址，此即 eth1 的第一个包；server2 的回应即 eth1 的第二个包

router 收到回应后，在ARP请求队列中找到缓存的ping包，将其发出，此即 eth1 的第三个包；接下来的几组ping包则是正常交互通信的结果

一段时间后， server2 维护自身ARP表，发起ARP确认表项状态，得到 router 的回应，此即 eth1的最后两个包